06

동적 계획법

그림으로 계산하고, 계산을 코드로 옮기기

1칸 계단: 오르는 방법은 하나



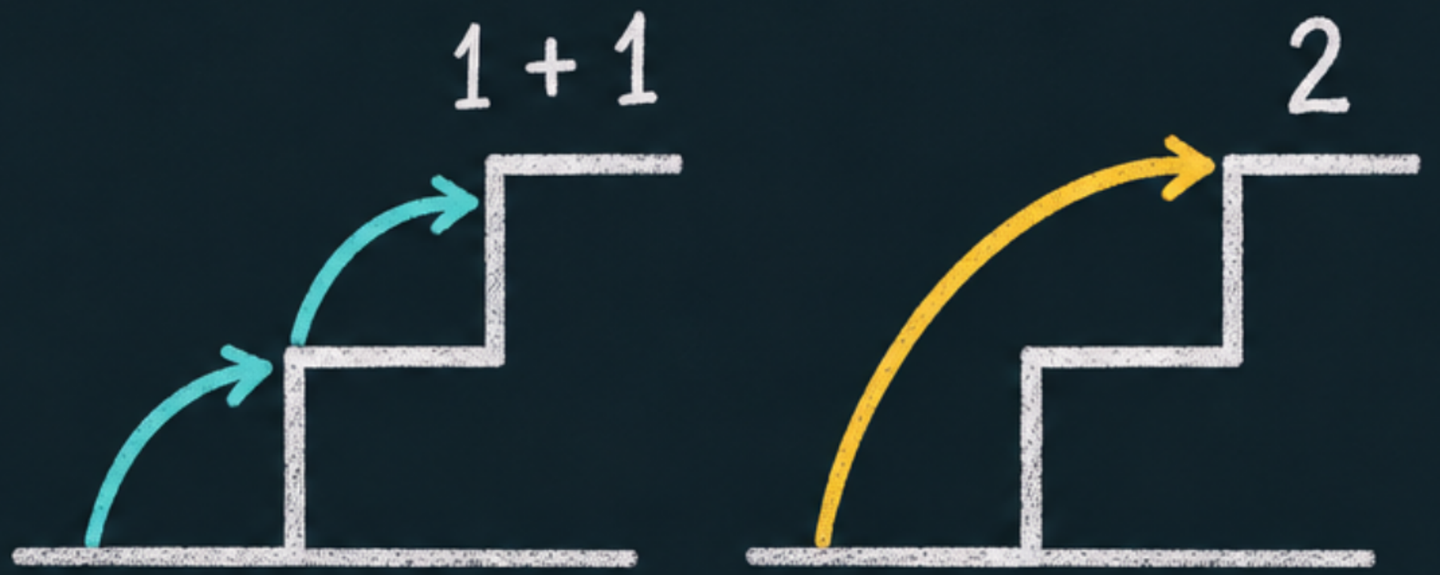
한 번에 1칸 또는 2칸

한 칸을 오르면 도착합니다.

꼭대기를 넘어가는 이동은 세지 않습니다.

1칸의 답 = 1

2칸 계단: 두 방법을 직접 셉니다



작은 계단은 나열해서 셉니다

1칸 + 1칸

한 번에 2칸

2칸의 답 = 2

3칸 계단: 순서가 다르면 다른 방법



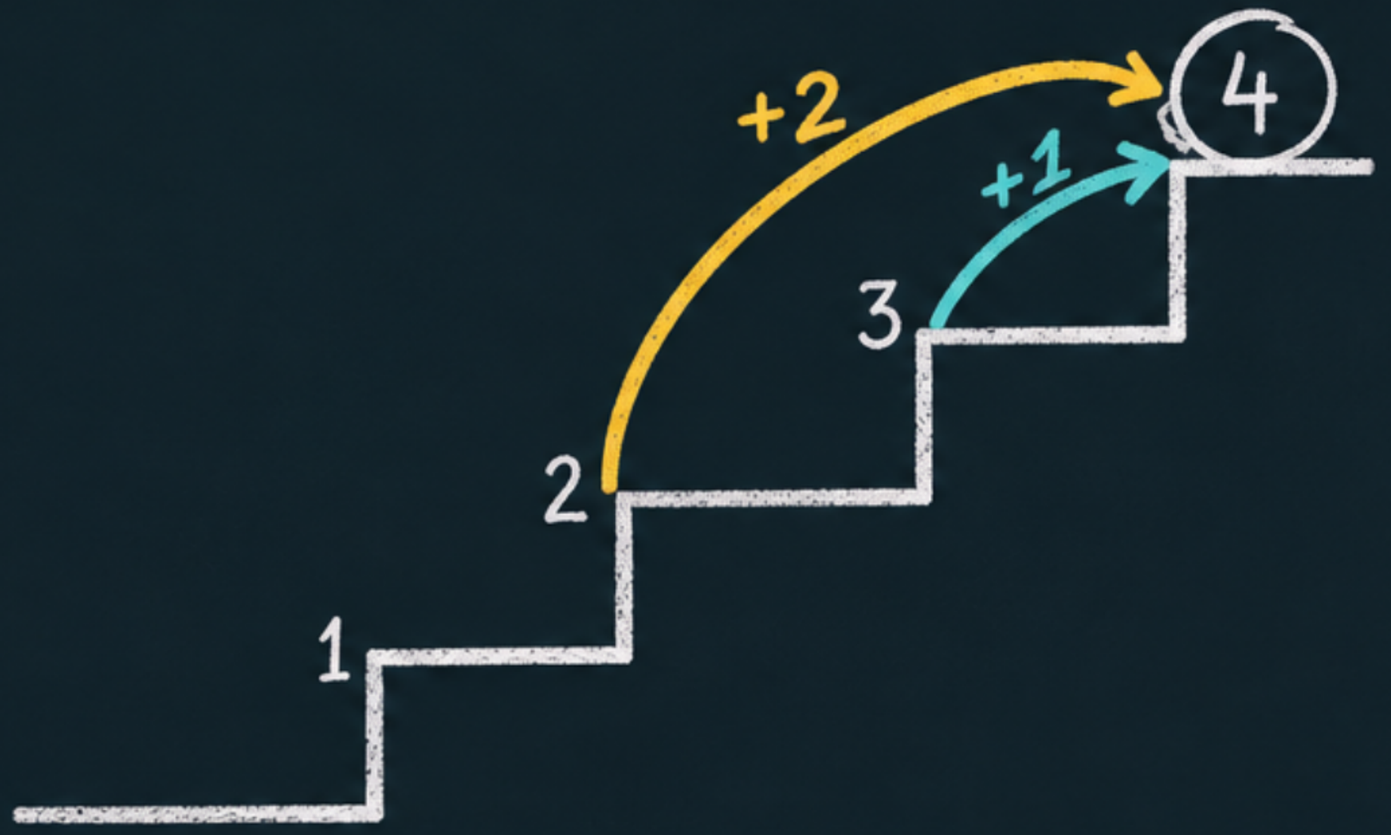
가운데와 오른쪽을 비교하세요

1 + 2와 2 + 1은 서로 다른 순서입니다.

그림의 경로를 한 줄씩 세면 됩니다.

3칸의 답 = 3

4칸에 도착하기 직전의 위치



마지막 이동만 거꾸로 봅니다

3칸에서 1칸 더 오르기

2칸에서 2칸 더 오르기

마지막 이동은 두 경우뿐

앞에서 센 답을 더하면 됩니다

3칸까지 3가지 + 마지막 1칸

2칸까지 2가지 + 마지막 2칸

3가지 + 2가지 = 5가지

두 묶음은 마지막 이동이 달라서 겹치지 않습니다.

이 계산을 짧게 적으면

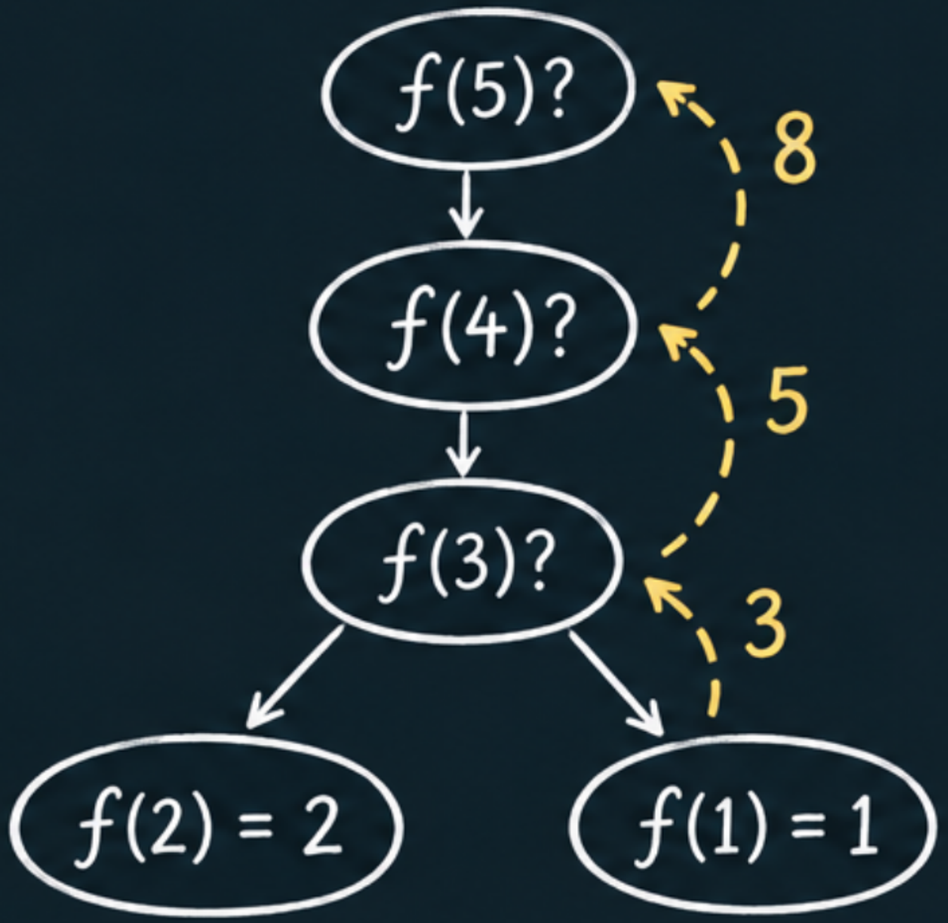
f(k): k칸까지 오르는 방법 수

f(1) = 1, f(2) = 2

f(4) = f(3) + f(2) = 5

3칸부터: f(n) = f(n−1) + f(n−2)

큰 답부터 물어보는 탑다운



원하는 높이부터 질문합니다

5칸의 답에 필요한 작은 답을 묻습니다.

1칸과 2칸은 이미 답을 압니다.

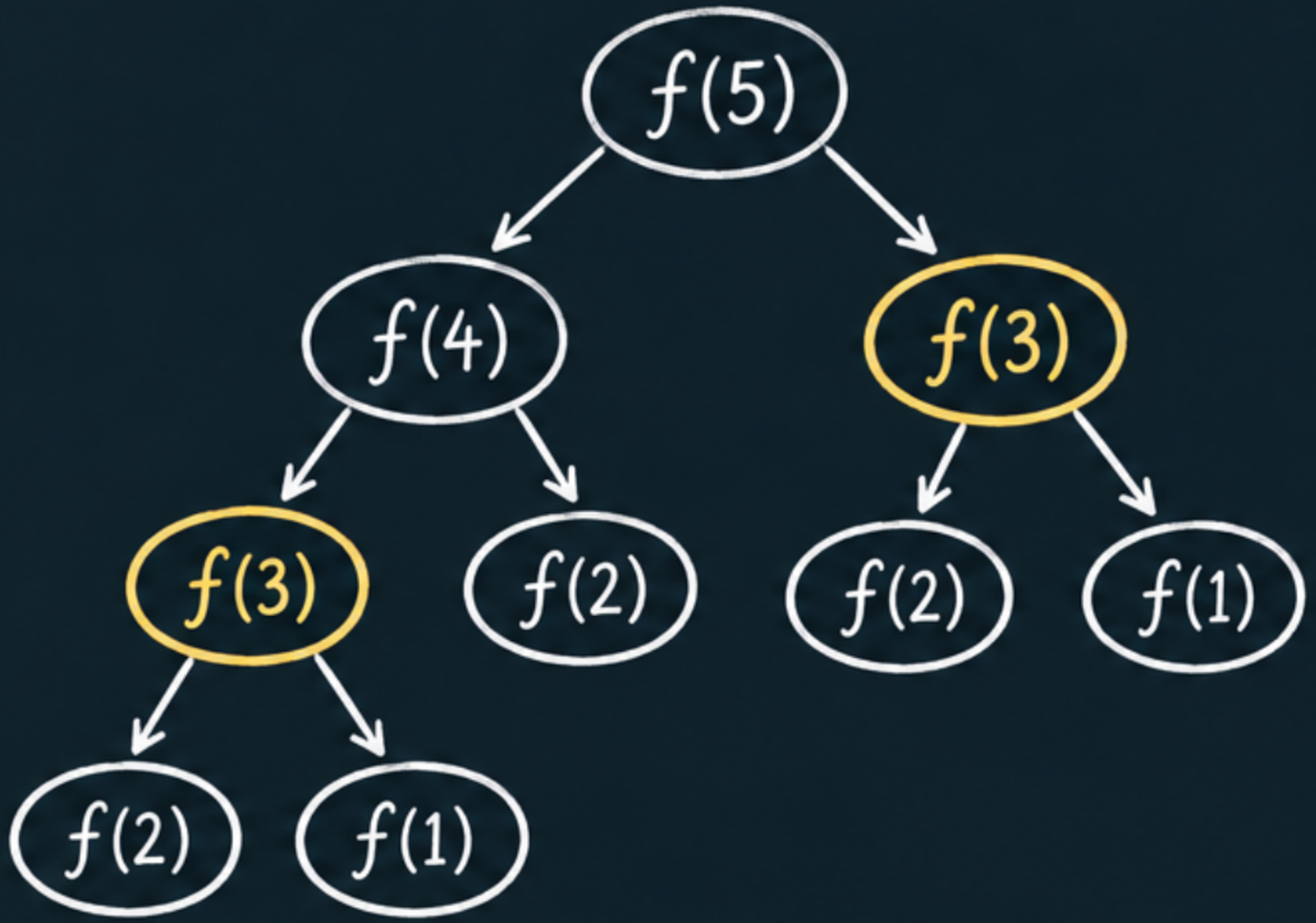
질문은 아래로, 답은 위로

우선 식을 그대로 코드로 옮기면

```
def f(k):
    if k <= 2:
        return k
    return f(k-1) + f(k-2)
```

1·2칸: 바로 반환 · 3칸 이상: 두 작은 답을 더함

같은 f(3)을 두 번 계산합니다



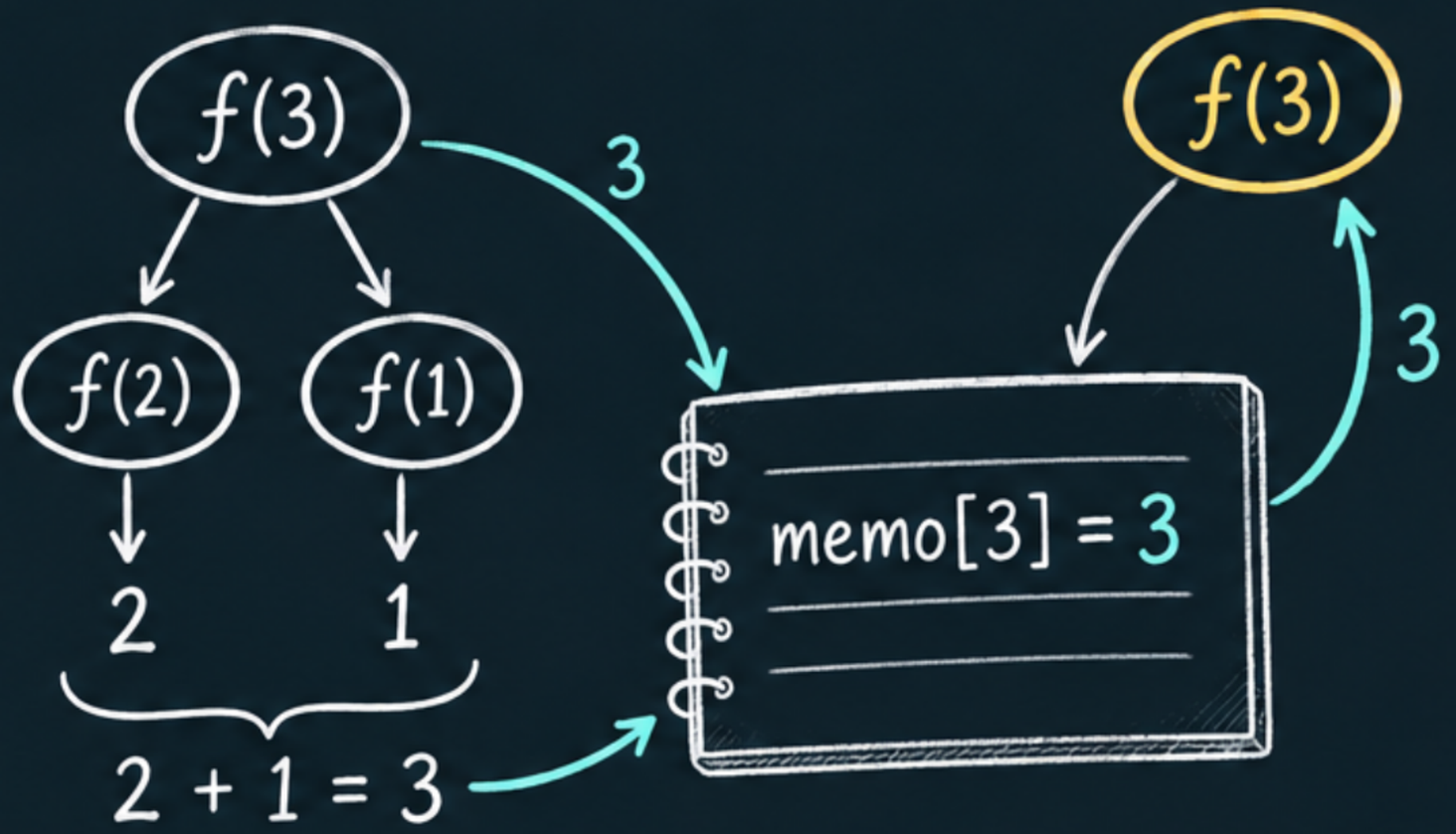
노란색 두 질문은 같습니다

입력이 둘 다 3이면 답도 둘 다 3입니다.

처음 얻은 답을 적어두면 재사용할 수 있습니다.

같은 계산을 다시 하지 않기

처음에는 계산하고, 다음에는 읽습니다



메모에는 높이와 답을 짝지어 적습니다

첫 요청: 2 + 1을 계산해 3을 저장

같은 요청: 저장한 3을 바로 반환

memo[3] = 3

Python 사전으로 메모하기

```
memo = {1: 1, 2: 2}

memo[3] = 3

3 in memo

memo[3]
```

1·2칸의 답을 적기 · 3칸의 답을 저장하기 · 저장 여부 확인하기 · 저장한 답 읽기

탑다운 전체 코드 · 비교용

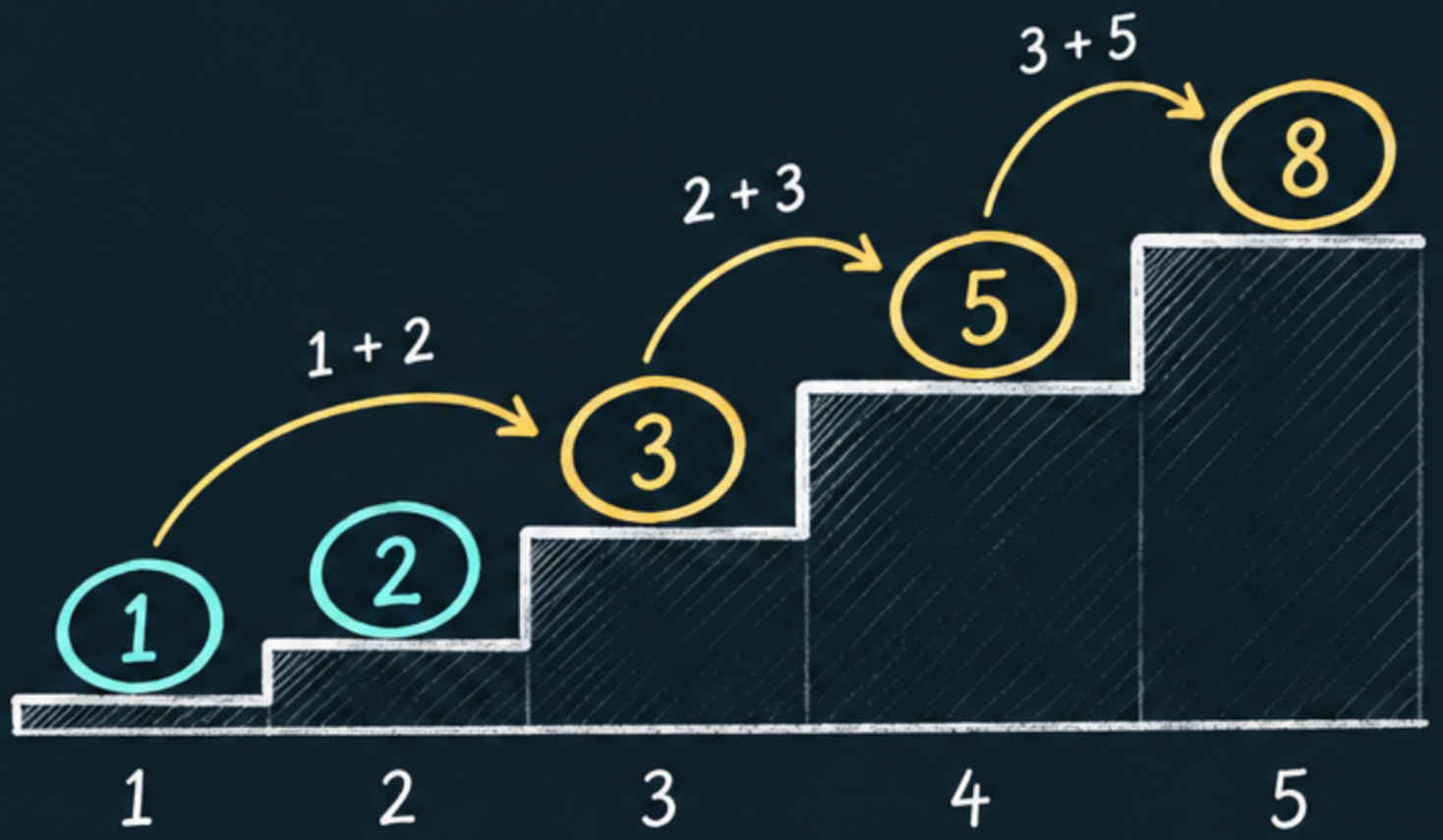
```
def climb_stairs(n):
    memo = {1: 1, 2: 2}

    def f(k):
        if k in memo:
            return memo[k]
        memo[k] = f(k-1) + f(k-2)
        return memo[k]

    return f(n)
```

memo는 한 번 생성 · f 호출끼리 공유 · 사이트 제출은 바텀업

작은 답부터 채우는 바텀업



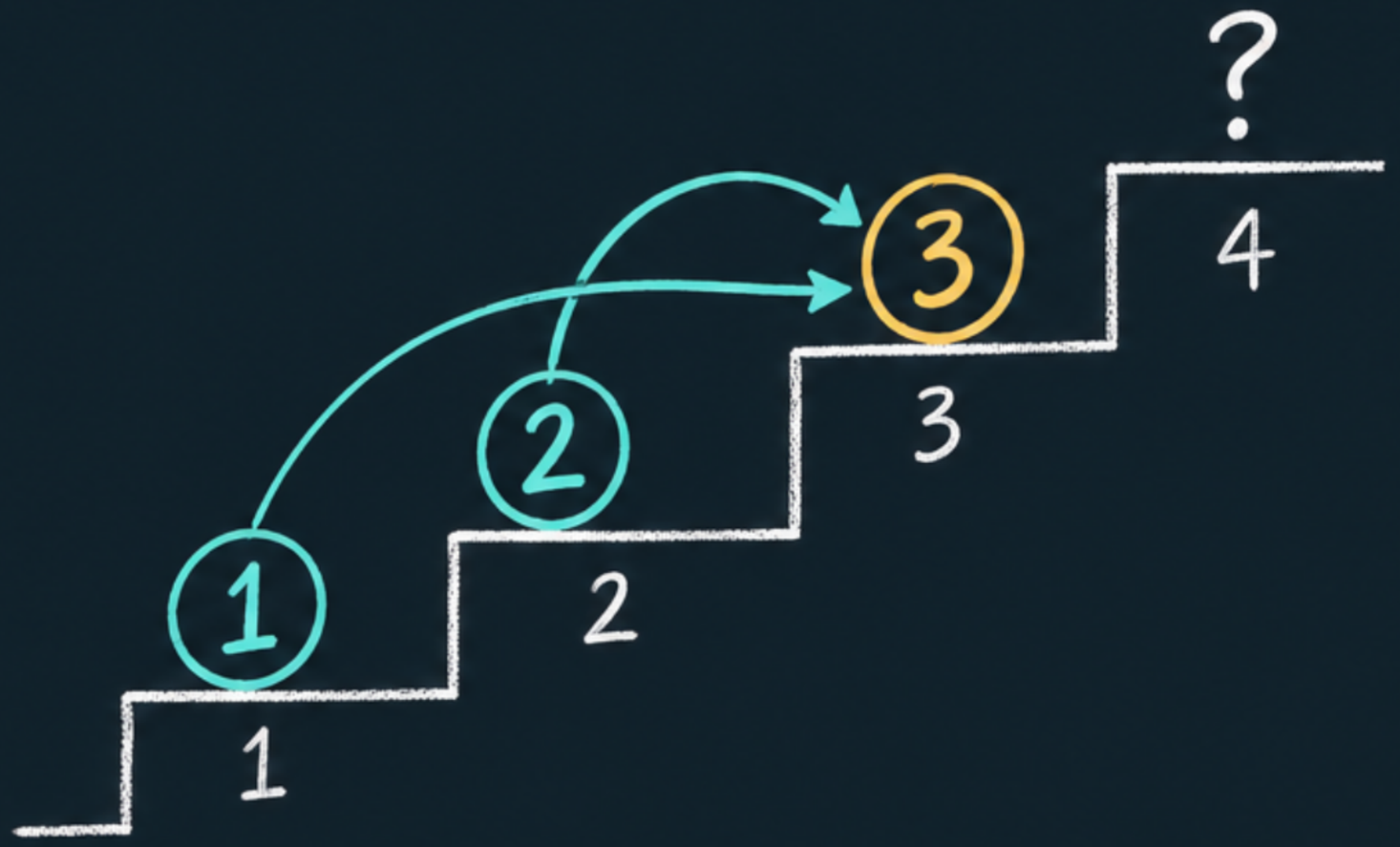
1과 2를 적고, 다음 답을 만듭니다

동그라미 안 숫자 = 방법 수

아래 작은 숫자 = 계단 높이

1, 2, 3, 5, 8

3칸의 답: 2 + 1



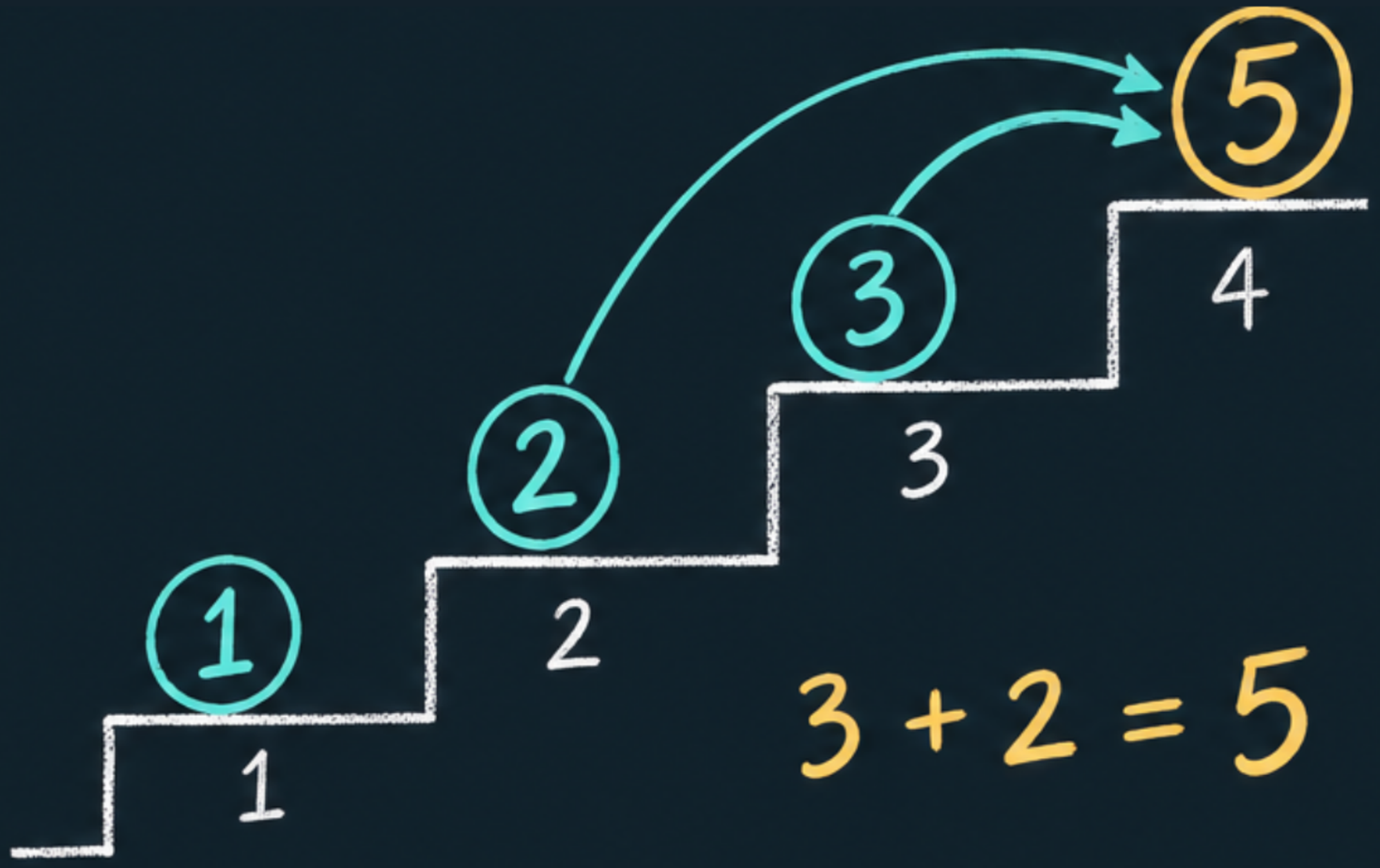
이미 적힌 두 숫자만 읽습니다

dp[2]에서 2를 읽고

dp[1]에서 1을 읽어 더합니다.

dp[3] = 2 + 1 = 3

4칸의 답: 방금 적은 3을 사용



새 답이 다음 계산의 재료가 됩니다

3칸의 답 3 + 2칸의 답 2

이제 4칸의 답은 5입니다.

dp[4] = 3 + 2 = 5

계단 오르기 · 바텀업 Python

```python
def climb_stairs(n):
    if n == 1:
        return 1
    dp = [0] * (n+1)
    dp[1] = 1
    dp[2] = 2
    for i in range(3, n+1):
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
```

n = 1: 바로 반환 · 1·2번: 시작값 · 3번부터: 앞의 두 답을 더함

반복문 한 줄을 그림처럼 읽으면

i = 3 → dp[3] = dp[2] + dp[1]

i = 4 → dp[4] = dp[3] + dp[2]

앞의 두 답을 읽어, 지금 칸에 적기

range(3, n+1)은 3, 4, …, n을 차례로 줍니다.

계단 오르기 · C

```c
long long climb_stairs(int n) {
    if (n == 1) return 1;
    long long dp[91] = {0};
    dp[1] = 1;
    dp[2] = 2;
    for (int i=3; i<=n; ++i) {
        dp[i] = dp[i-1] + dp[i-2];
    }
    return dp[n];
}
```

int n: 계단 높이 · long long: 방법 수 · 계산 순서는 Python과 같음

높이는 작아도 방법 수는 큽니다

46칸의 방법 수: 2,971,215,073

32비트 int 최댓값: 2,147,483,647

방법 수는 long long에 저장

계단 높이 n과 반복 번호 i는 int로 충분합니다.

두 접근이 출발하는 곳

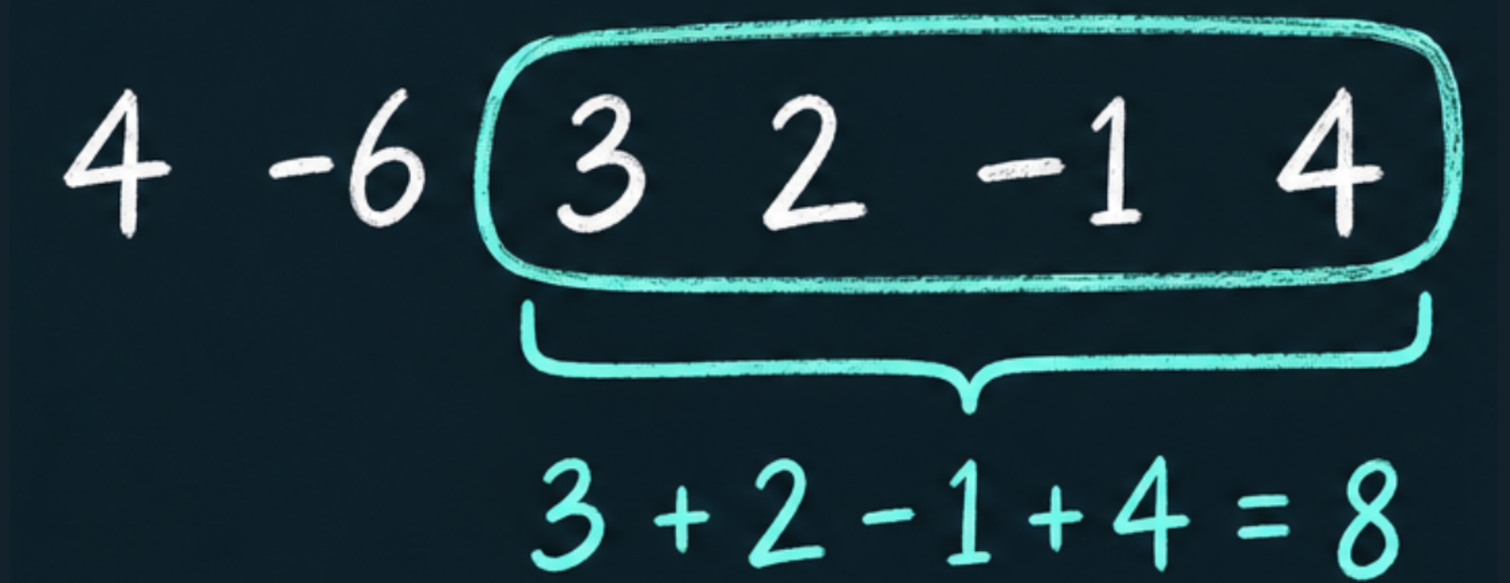
탑다운: 원하는 답에서 작은 질문으로

바텀업: 아는 작은 답에서 다음 답으로

둘 다 앞서 구한 답을 저장해 사용

사이트의 계단 오르기는 바텀업으로 작성합니다.

연속 부분합: 붙어 있는 숫자 고르기



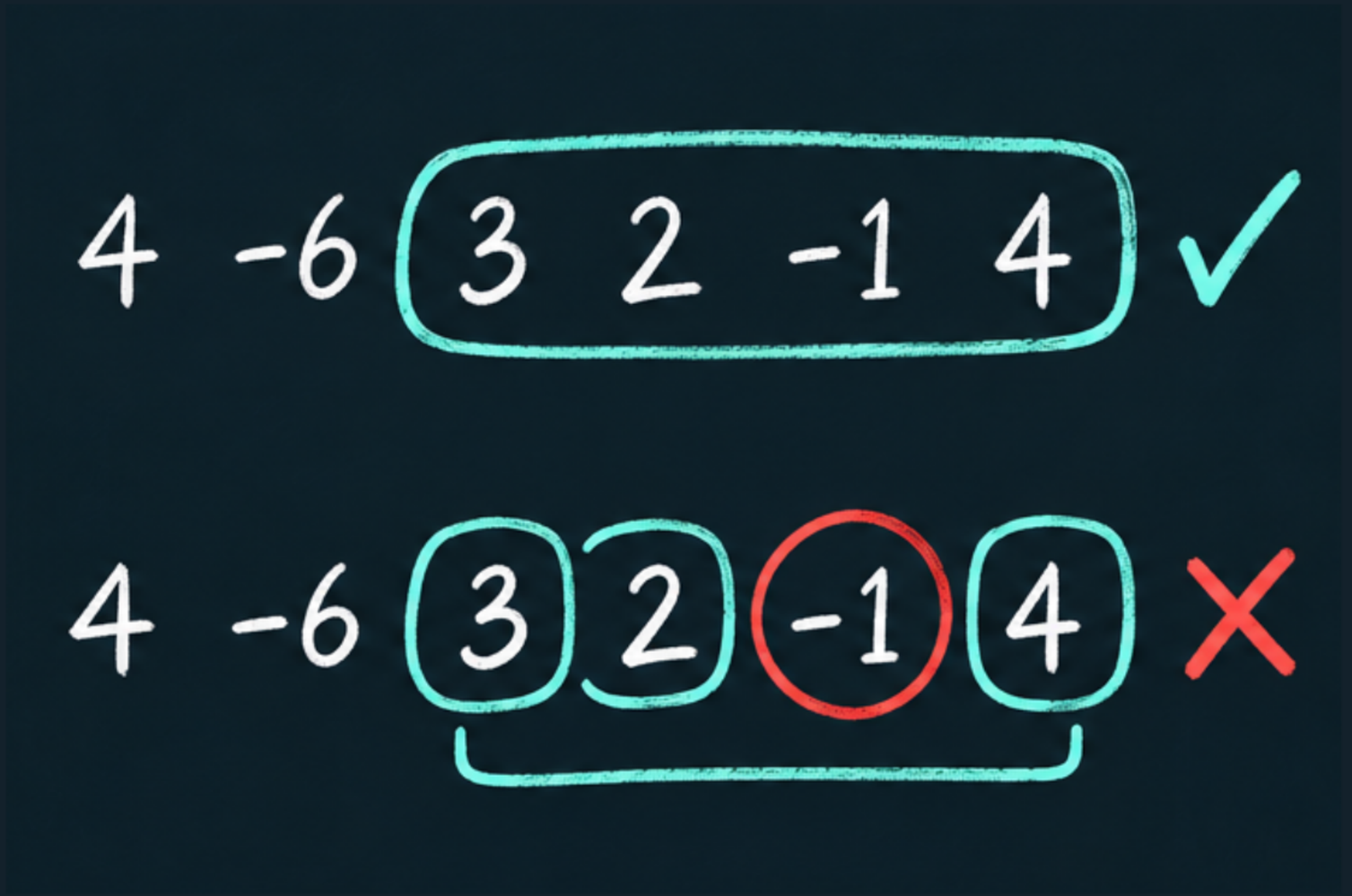
한 구간을 골라 합을 크게 만듭니다

3부터 마지막 4까지 고르면

3 + 2 – 1 + 4 = 8

반환할 값은 합 하나

가운데 -1을 건너뛸 수 없습니다



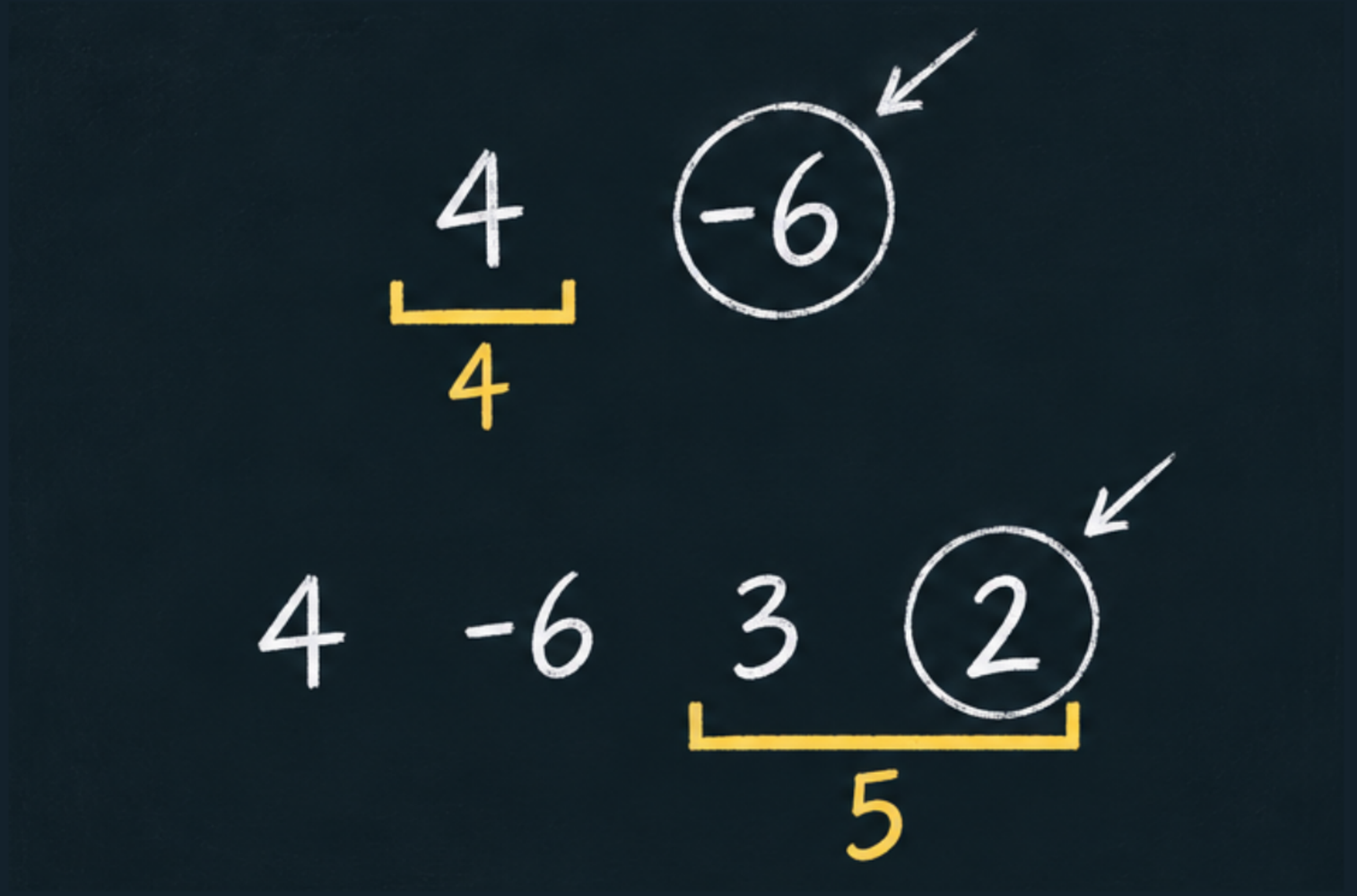
시작부터 끝까지 모두 포함

위: 연속한 구간이라 가능

아래: 중간 숫자를 빼서 불가능

최소 한 숫자는 골라야 합니다.

숫자가 추가되면 최고 구간은?



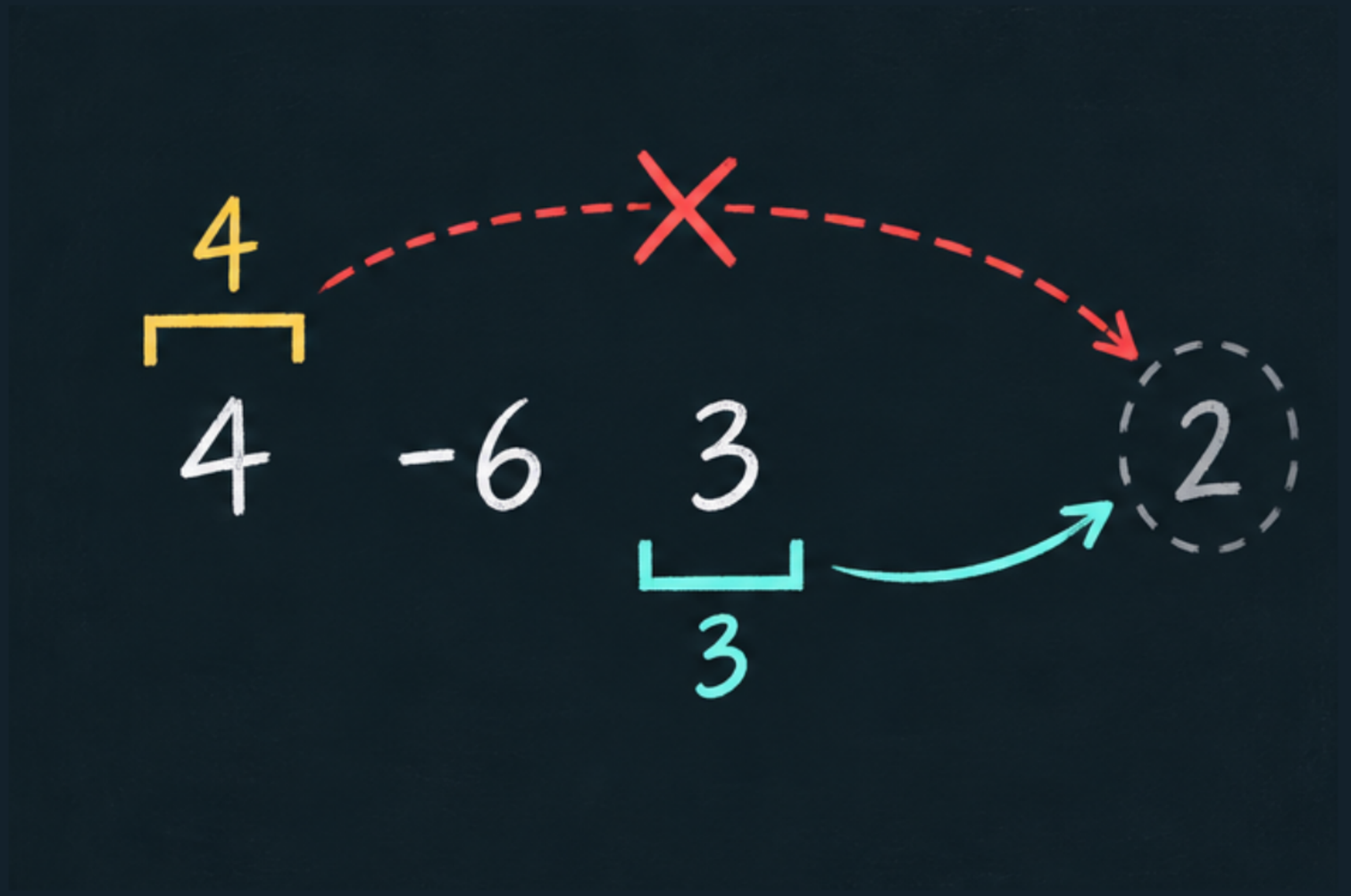
**이전 최고를 유지하거나,
새 구간으로 바뀝니다**

새 최고가 생긴다면

이번 숫자를 반드시
포함합니다.

**이번 숫자에서 끝나는
구간에 주목**

이어 붙일 수 있는 답을 기억해야 합니다



노란색과 초록색은 다른 답입니다

노란색: 지금까지 전체 최고

초록색: 현재 위치에서 끝나는 최고

다음 숫자에는 초록색 답을 붙입니다.

첫 숫자 4만 읽었을 때

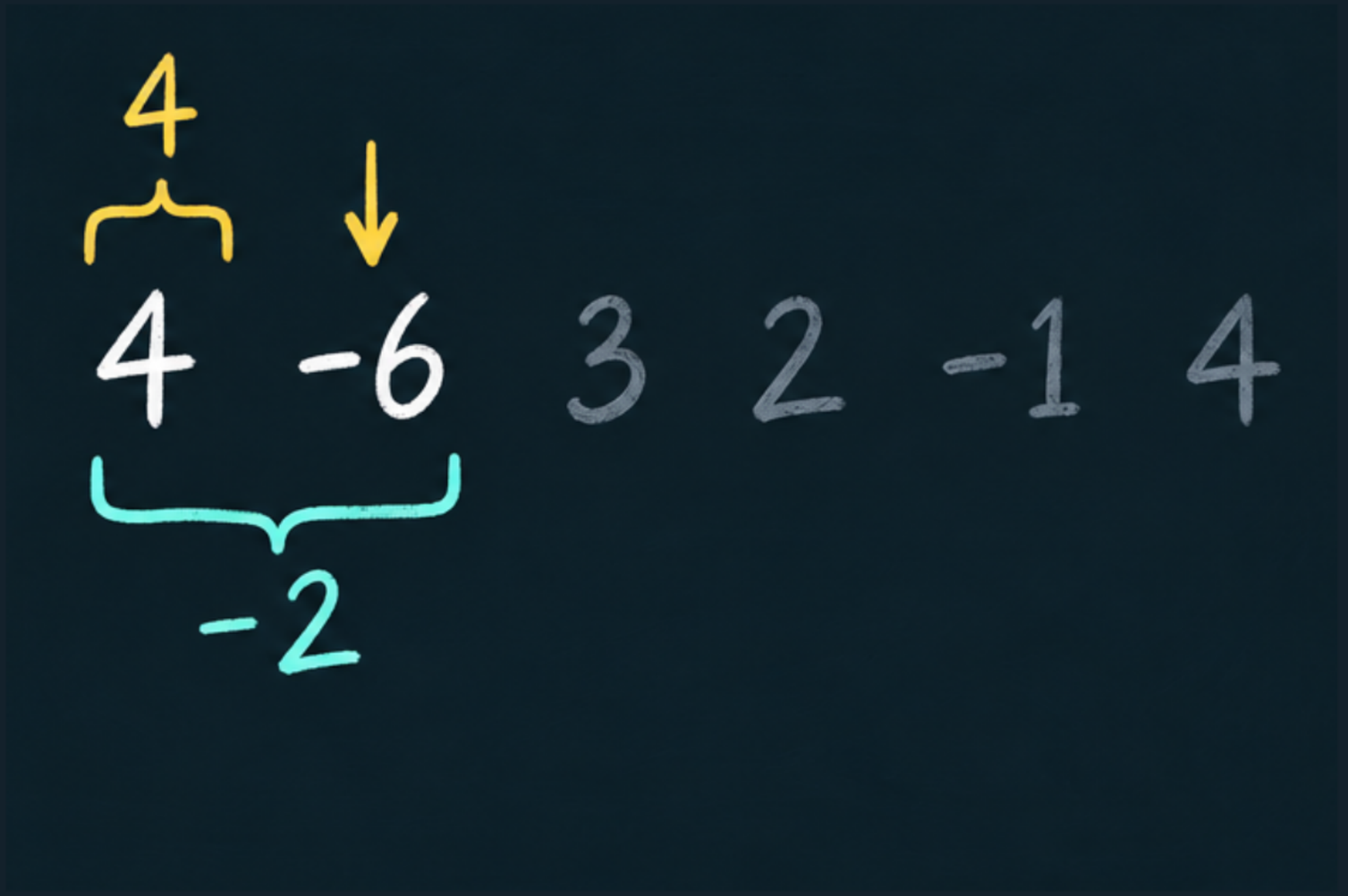
지금 읽은 숫자: 4

만들 수 있는 구간: [4]

현재 끝나는 합 4 · 전체 최고 4

두 기록 모두 첫 숫자로 시작합니다.

-6을 붙이면 현재 합은 -2



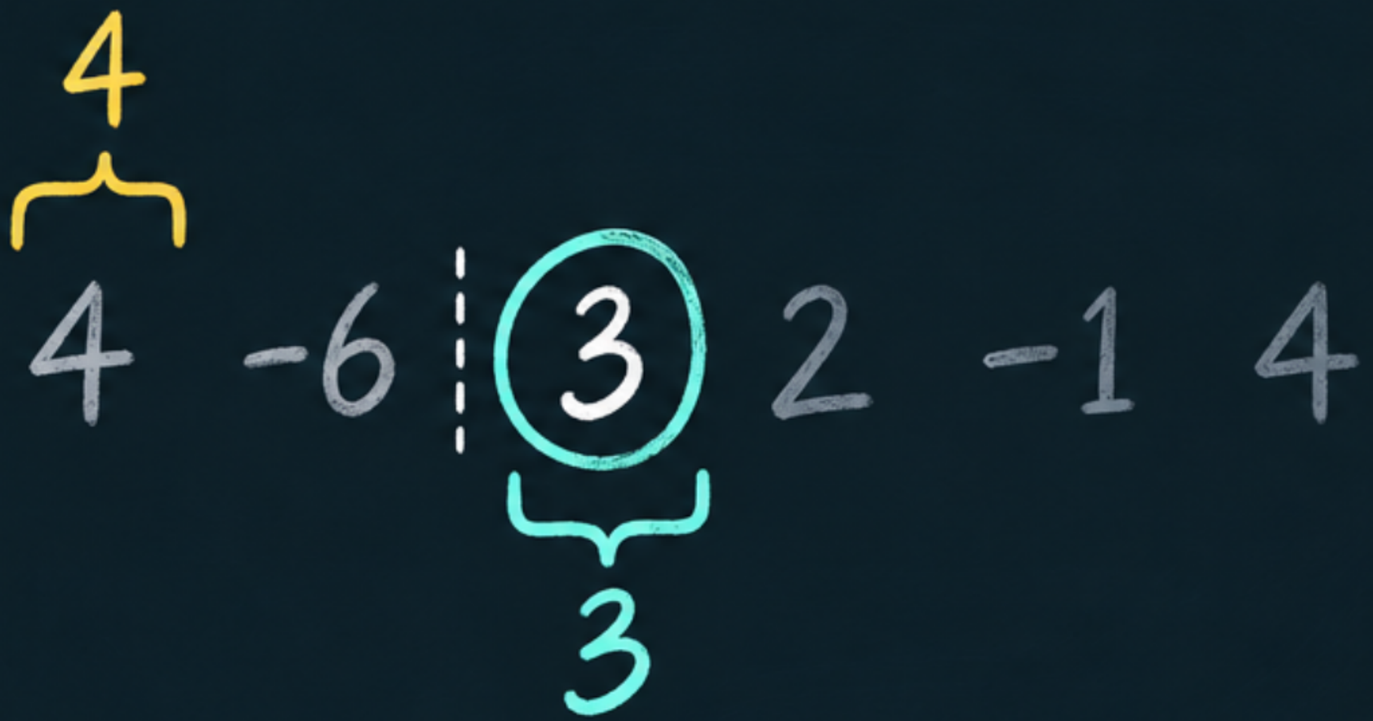
이어 붙임 -2와 새 시작 -6 중 선택

현재 끝나는 합: -2

전체 최고는 이전 4를 유지

현재 합이 줄어도 최고 기록은 남습니다.

3에서는 새로 시작하는 편이 좋습니다



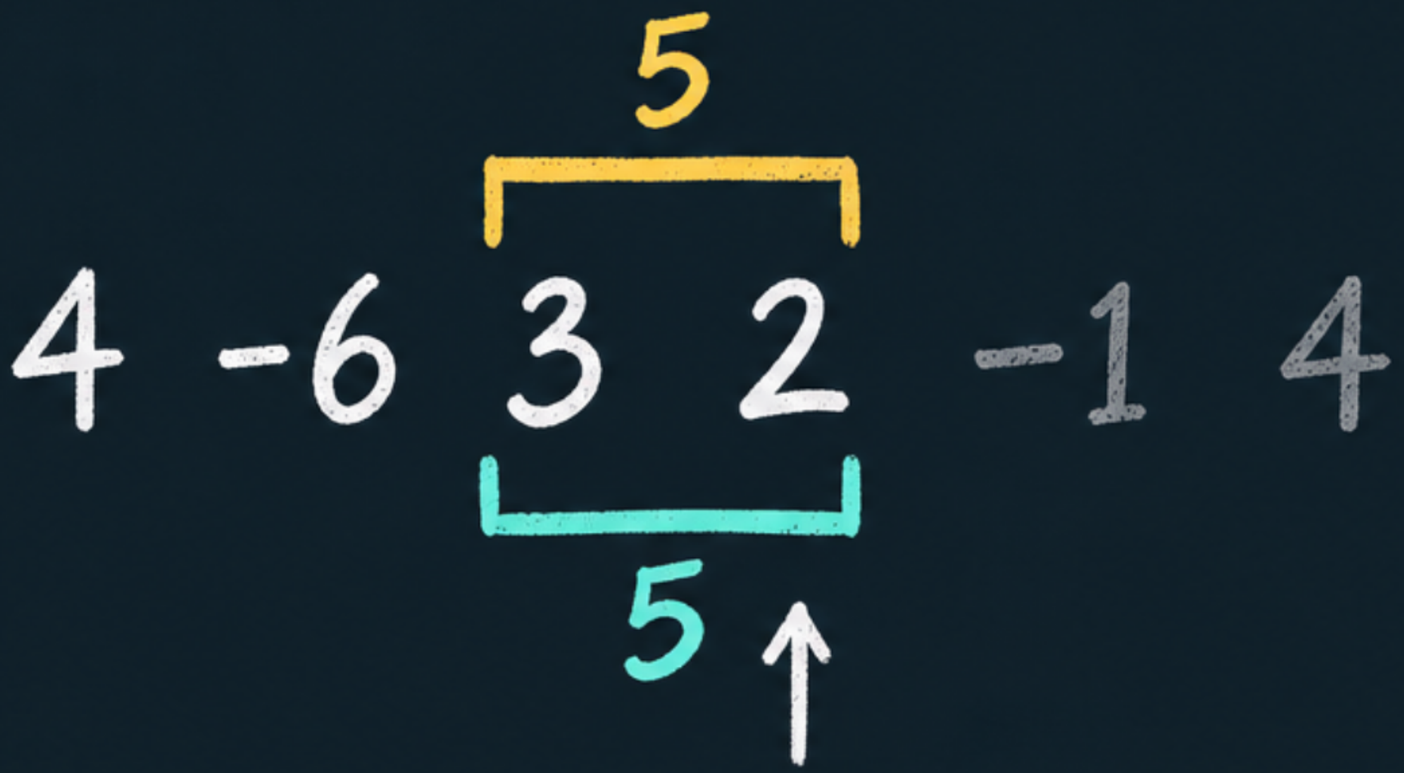
앞의 -2를 붙이면 1,
혼자 시작하면 3

3 하나로 새 구간을 시작합니다.

전체 최고는 아직 4입니다.

현재 끝나는 합 = 3

2를 붙이면 합이 5



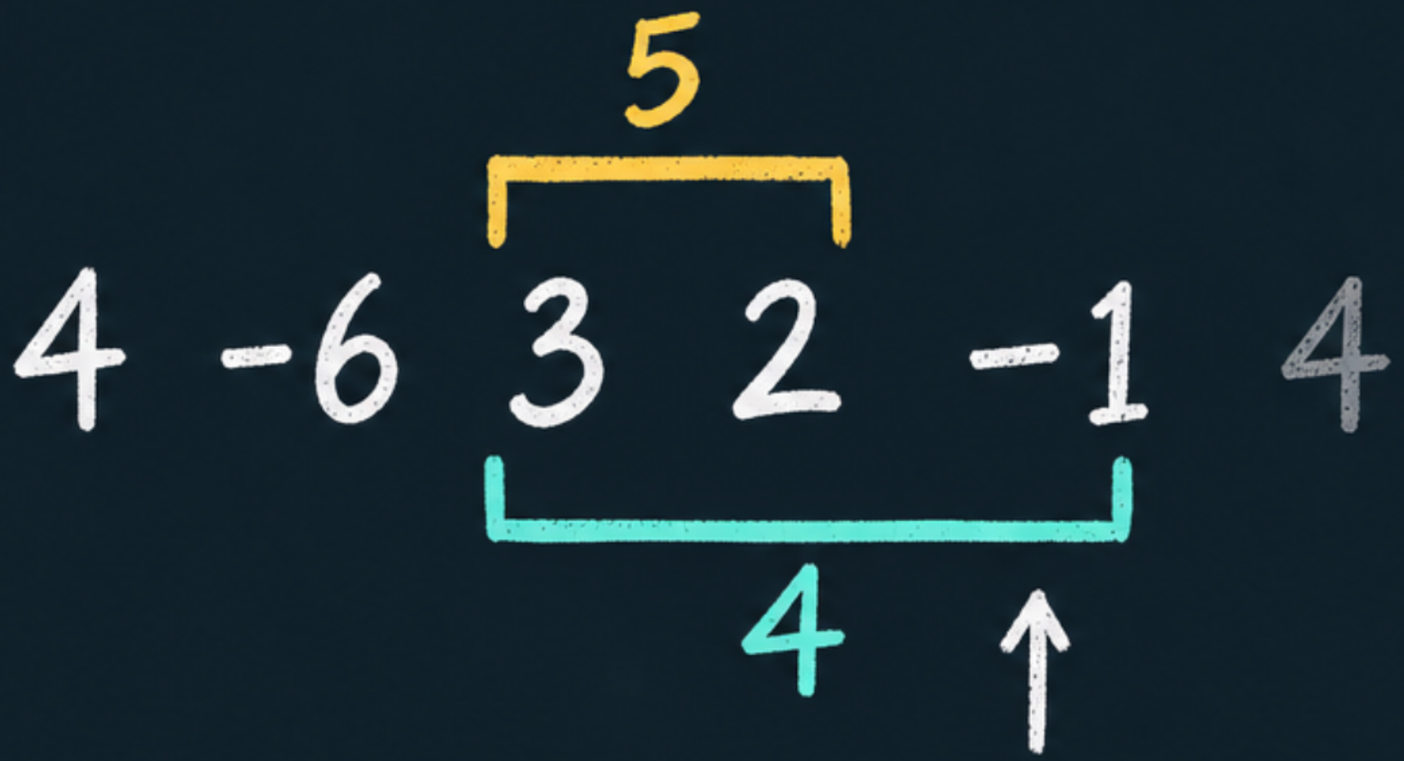
3 + 2가 2 하나보다 큽니다

초록색 합은 5

이전 최고 4보다 커서 노란색도 5

새 최고 구간은 [3, 2]

-1을 읽으면 두 기록이 달라집니다



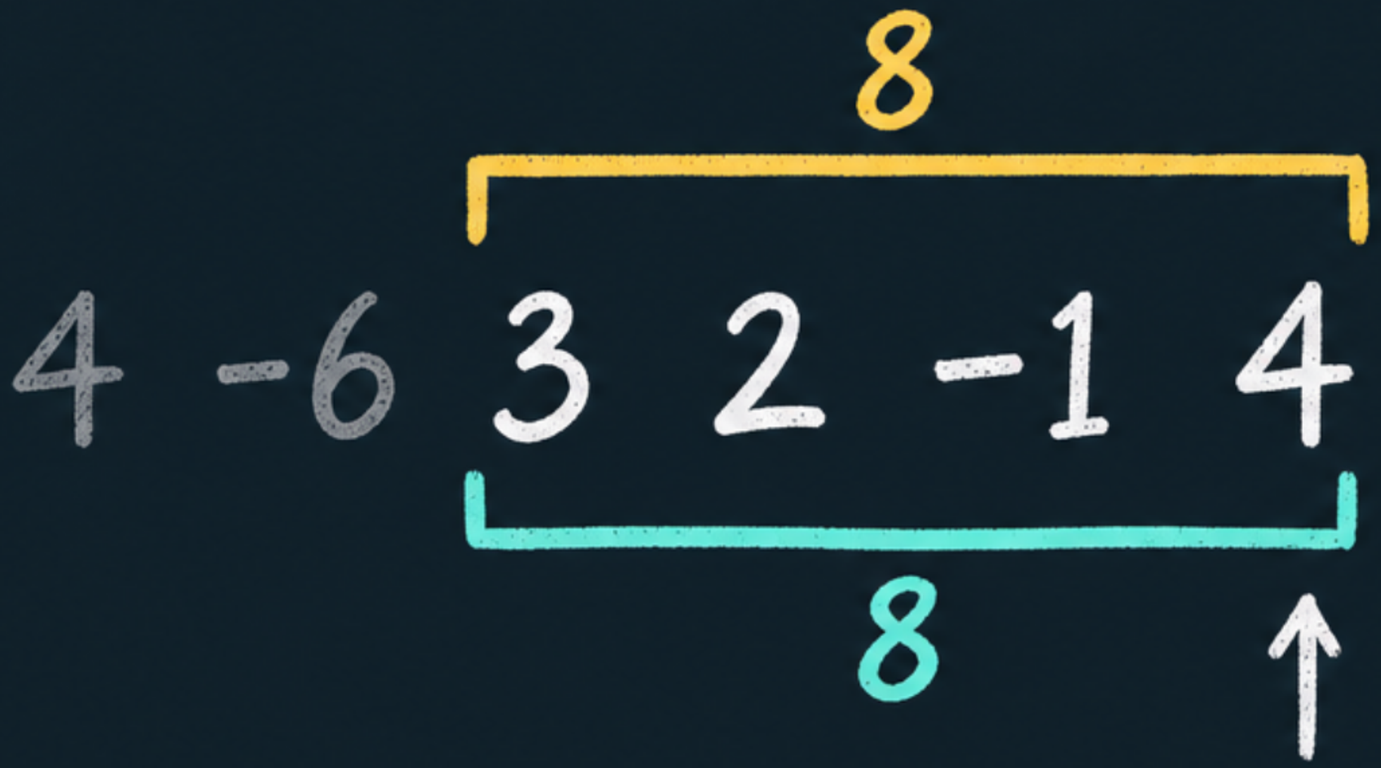
초록색은 이번 숫자까지 포함합니다

[3, 2, -1]의 합 = 4

전체 최고 [3, 2]의 합 = 5

현재 끝나는 합 4 · 전체 최고 5

마지막 4를 붙이면 합이 8



앞의 합 4에 이번 숫자 4를 더합니다

이어 붙임 8과 새 시작 4 중 8

이전 최고 5와 비교해도 8

최종 답 = 8

두 기록의 이름: ending과 best

ending: 지금 숫자에서 끝나는 최고 합

best: 지금까지 전체에서의 최고 합

ending을 구한 뒤 best와 비교

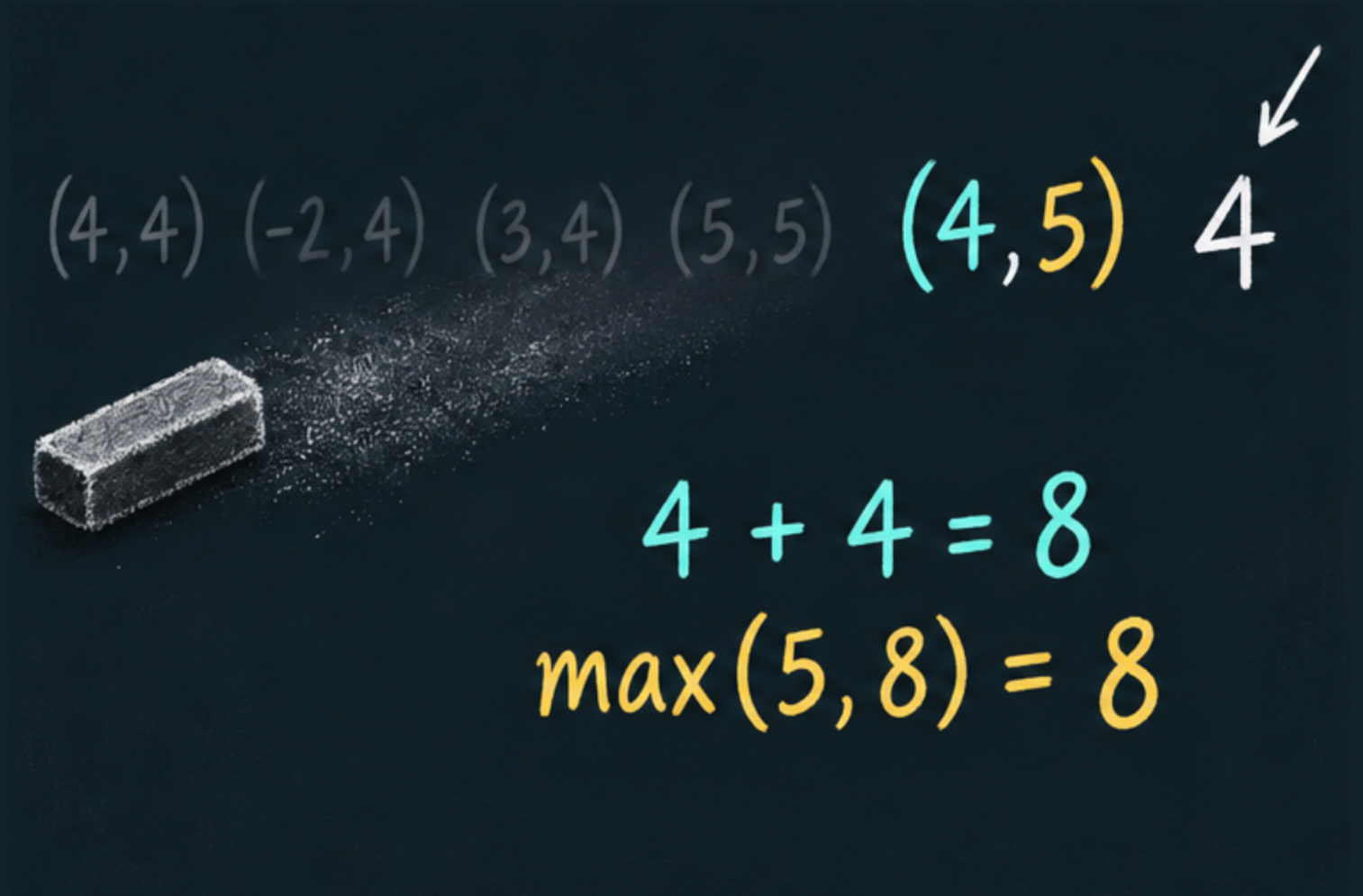
ending = max(x, 이전 ending + x)
best = max(이전 best, 새 ending)

먼저 계산한 답을 모두 적어두면

```
1   def max_subarray(a):
2       ending = [0] * len(a)
3       best = [0] * len(a)
4       ending[0] = best[0] = a[0]
5       for i in range(1, len(a)):
6           ending[i] = max(a[i], ending[i-1] + a[i])
7           best[i] = max(best[i-1], ending[i])
8       return best[-1]
```

초록색 기록: ending[i] · 노란색 기록: best[i]

다음 계산에는 마지막 두 값만 필요



회색 기록은 다음 계산에서 읽지 않습니다

남길 값: ending = 4, best = 5

이번 4를 읽고 두 값을 8로 갱신

배열 대신 변수 두 개

연속 부분합 · Python 전체 코드

```python
def max_subarray(a):
    ending = a[0]
    best = a[0]
    for i in range(1, len(a)):
        ending = max(a[i], ending + a[i])
        best = max(best, ending)
    return best
```

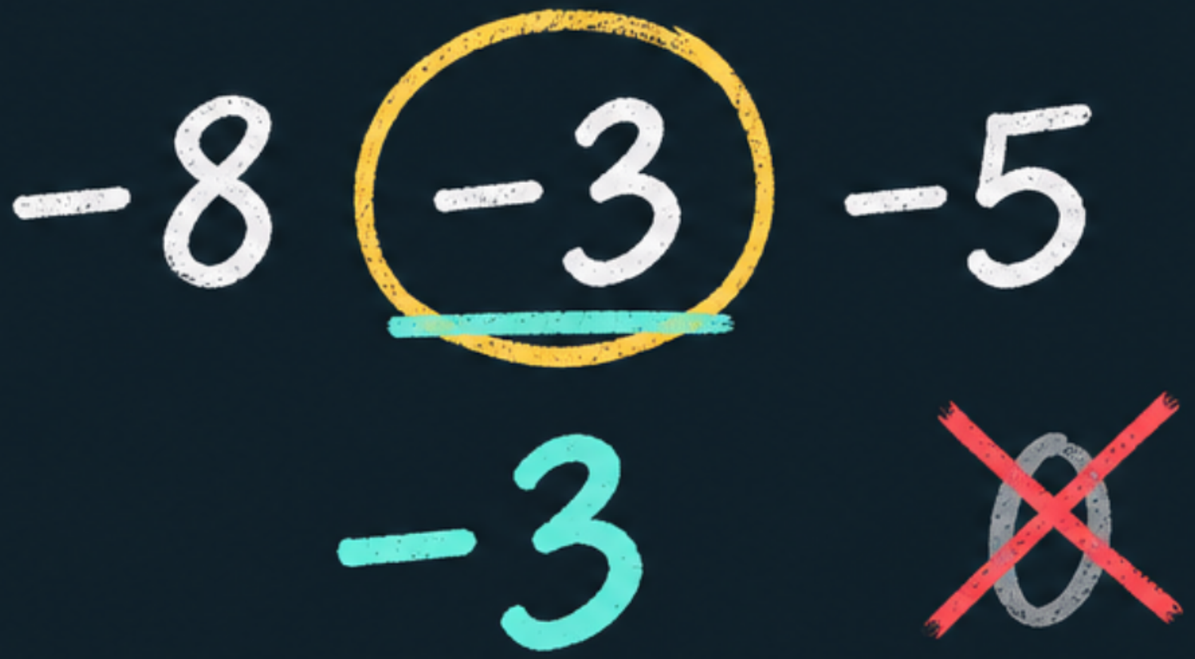
먼저 ending 갱신 · 그다음 best 갱신 · 반환값은 best

연속 부분합 · C

```c
long long max_subarray(const int a[], int n) {
    long long ending=a[0], best=a[0];
    for (int i=1; i<n; ++i) {
        long long extend=ending+a[i];
        if (extend>a[i]) ending=extend;
        else ending=a[i];
        if (ending>best) best=ending;
    }
    return best;
}
```

a: 입력 숫자 배열 · n: 원소 수 · 합은 long long

모두 음수여도 하나는 골라야 합니다



빈 선택은 허용하지 않습니다

셋 중 가장 큰 –3 하나를 고릅니다.

시작값을 무조건 0으로 두면 틀립니다.

반환값 = –3

다음 계산에 필요한 답을 저장합니다

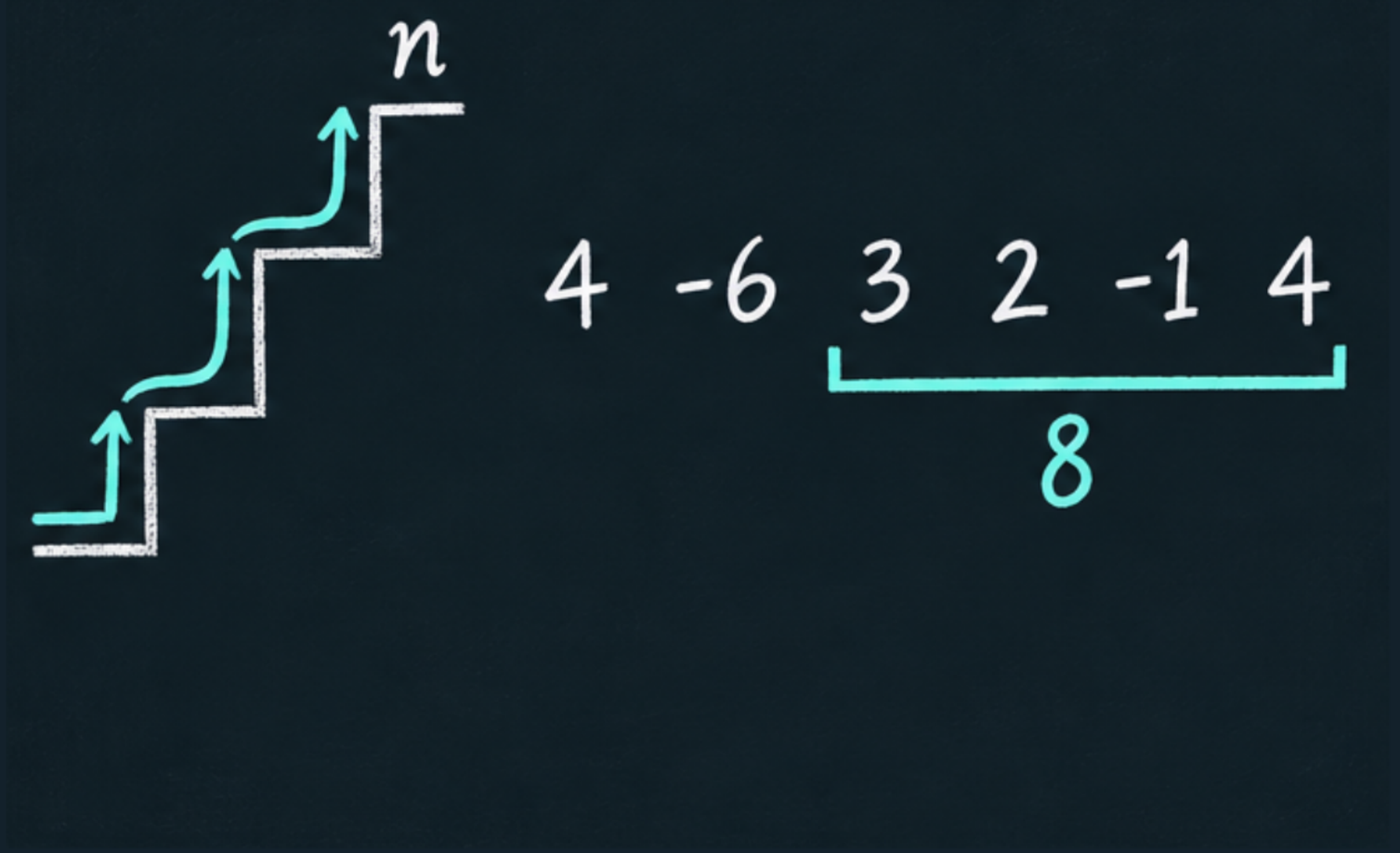
계단: 앞의 두 높이의 방법 수

연속 부분합: ending과 best

다음 답에 필요한 정보를 남기기

계산을 다시 하기보다, 이미 구한 답을 사용합니다.

두 문제를 직접 구현해 봅시다



6-2 계단 오르기 · 6-3 연속 부분합

climb_stairs(4) → 5

max_subarray([4,−6,3,2,−1,4])
→ 8

그림에서 한 계산을 코드로 옮기기